Critiques of research papers

Shubham Jha

February 2018

1 Dynamic Routing between Capsules

Sara Sabour Nicholas Frosst Geoffrey E. Hinton

1.1 Introduction and methodology

Convolutional neural networks are very popular in the field of Computer Vision.They have demonstrated incredible performance in areas like object detection, segmentation etc. CNNs use parameter sharing and local connectivity to keep the number of parameters low, which makes training and inference using CNNs tractable. While initially intended for images, they are now being in the fields of natural processing and time series analysis too.

One of the major drawbacks of CNNs is the use of max pool layer. Max pooling consists of sliding a window of fixed size across the activation map of the previous layer and only passing the maximum activation. Geoffrey Hinton has vehemently opposed the use of the max pooling, and called it a 'big mistake'and also stated that 'the fact that it works so well is a disaster'. The reason for such opposition is that max pool loses a lot of information. Due to the use of max pool layers, CNNs are invariant to translation and slight rotation but not equivariant to it. The neural activities of capsules vary as viewpoint varies. This is in contrast to normalization methods such as spatial transformer networks which eliminate viewpoint variation from the activities.

This paper presents a novel architecture which is composed of capsules. A capsule is a group of neurons whose activity vector encodes the instantiation parameters of an entity. The entity may be an object or a part of an object. Instantiation parameters of an object are basically its properties such as pose, velocity, albedo etc. The length of the vector represents the probability that an object is present in the image and the orientation of the vector represents the instantiation parameters.

Capsule nets, like CNNs, represent information in a hierarchical form. They are based on the concept of inverse graphics. Every entity is made of several smaller entities oriented in space in some way and placed at some specific position in space. The authors present a novel training algorithm called Routing-by-agreement to train the capsules. The lower level capsules try to predict the output of higher level capsules and the capsules which have a greater agreement with the higher layer capsules are coupled to the parent even more through a positive feedback loop.

Since we are dealing with vectors here, as compared to CNNs where we deal with scalar inputs and scalar outputs, we can't use the traditional activation functions used with CNNs. Hence, the authors have used a novel activation function of the form-

$$v_j = \frac{||s_j||^2}{1 + ||s_j||^2} \frac{s_j}{||s_j||}$$

This activation function scales the length of all the vectors in the $[0-1]$ range. Every capsule in a layer is coupled to capsules in a higher layer through coupling coefficients, which are scalar weights. The coupling coefficients represent the amount of coupling between a particular capsule and its parent in the next layer and are adjusted using the routing-by-agreement mechanism. Capsules having a higher level of agreement with a capsule in the next layer will have a larger value for the coupling coefficient. The coupling coefficients also represent the probability that the output of a capsule is routed to a parent in the next layer and their values sum up to 1. The weights W represent transformation matrices, which are used to represent affine transformations in space. According to the authors the algorithm converges in about $r = 3$ iterations. Running the algorithm for a larger number of iterations results in overfitting.

The authors use the following loss function for training

$$L_k = T_k \ max(0, m^+ - ||v_k||)^2 + \lambda(1 - T_k) \ max(0, ||v_k|| - m^-)^2$$

where $m^+ = 0.9$ and $m^- = 0.1$ and $T_k = 1$ iff a digit of class k is present.

The authors present a basic CapsNet architecture consisting of two convolutional layers and one fully connected layer. The authors refer to the first two convolutional layers as- Convolutional Layer and PrimaryCaps Layer respectively, and the fully connected layer as DigitCaps layer. The DigitCaps layer has one 16D capsule per digit class, which means that it outputs a vector of length 16 for each of the 10 classes. The authors have also used a three-layer reconstruction subnetwork which takes as input a 16D vector from the DigitCaps layer and tries to reconstruct the original image. This reconstructed image is used to add an extra 'reconstruction loss' term to the loss function. This extra term acts as a regularizer. This architecture, combined with the routing-by-agreement mechanism produces impressive results.

The authors tested the CapsNet on the MNIST dataset and achieved an error rate of of 0.25%. The authors also trained a baseline CNN with 3 convolutional layers and two fully connected layers. They used dropout and the cross entropy cost function to train the network. This CNN achieved an error rate of 0.39%. The baseline CNN contained 35.4M parameters while the CapsNet contained only 8.2M parameters(reconstruction subnetwork included). Performance similar to the CapsNet has only been achieved with much deeper networks, and never with a network as shallow as the CapsNet even though the authors performed minimal dataset augmentation, wherein the images were shifted by up to 2 pixels in each direction with zero padding.

The authors also tested the CapsNet on two contrived datasets called MultiMNIST and affNIST. MultiMNIST was created by overlaying a digit on top of another digit. On this task, the CapsNet achieves similar performance(5% classification error rate) to the sequential attention model which was the previous state of the art. But the sequential attention model was tested on a dataset which the digits had very less overlap ($<$4%) as compared to MultiMNIST(80% overlap). Using the MultiMNIST dataset, the authors also demonstrate the ability of CapsNet to segment the two digits in an image. The authors also test the CapsNet on the affNIST dataset to test it's robustness to small affine transformations. This dataset contains MNIST digits with random small affine transformation. The CapsNet achieved 79% accuracy on this dataset, while a CNN with similar nummber of parameters as the CapsNet achieved only 66% accuracy. The CapsNet and CNN weren't trained on this dataset.

The authors performed analysis in which the outputs of the DigitCaps layer were perturbed slightly and passed to the reconstruction subnetwork. This analysis proved that the elements of the output vector of the DigitCaps layer corresponded to instantiation parameters of the digits such as pose, stroke width etc.

1.2 Shortcomings

This work, though impressive, isn't perfect. The authors haven't stated in the paper how the weights W are learned. It's very likely that the weights W are learned using standard backpropagation algorithm. Also, the routing-by-agreement algorithm introduces an additional hyperparameter r (number of iterations for the routing-by-agreement mechanism). Although the algorithm shows an impressive performance on the MNIST dataset, it fails to perform so well on the CIFAR-10 dataset, achieving an error of 10.6%. This is much higher than the current state of the art (3.47%). The authors account for this reduction in performance by reasoning that the CapsNet tries to account for everything present in an image, and in the CIFAR-10 images, the backgrounds are too varied, which the CapsNet isn't able to model well.

1.3 Directions for future work

More work needs do be done to make CapsNets generalize to more complex datasets such as CIFAR-10, which contain background clutter. Also, since CapsNets extract pose information from images, they can be very effective at 3D reconstruction from images, and hence the application of CapsNet in this field needs to be explored. The ability of CapsNet to extract pose can also be quite useful in the domain of robotics for tasks such as grasping.

2 Human-level control through deep reinforcement learning

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg & Demis Hassabis

2.1 Introduction and methodology

Reinforcement Learning is an area of machine learning concerned with how agents ought to take actions in an environment so as to maximize some notion of cumulative reward. In 2015, a paper called 'Human-level control through deep reinforcement learning' was published in the Nature journal which aroused a lot of excitement in the Reinforcement Learning community. The authors of the paper believed that a parallel existed between the concept of RL and how animals solve the RL problem. A chemical called Dopamine constitutes the reward mechanism in humans and this mechanism is a part of a system that closely resembles Temporal Difference RL. Moreover, humans and animals receive rich sensory inputs from their environments and observe a large state. Previous RL algorithms were unable to cope with such large state spaces, and hence, their applicability had been limited to domains where useful features could be handcrafted, or to domains with fully observed low-dimensional state spaces. In this work, the authors leverage the recent advances in the field of deep learning to develop and artificial agent, termed a deep Q-network, that learns policies directly from high dimensional sensory inputs using end-to-end reinforcement learning. The agent was tested Atari 2600 games and it achieved human level performance on 49 games. The only input to the agent was the pixel information from the game screen, which is similar to the input a human would receive while playing the game. The network used the same architecture and hyperparameters for all the games. This work bridges the divide between high-dimensional sensory inputs and actions, resulting in the first artificial agent that is capable of learning to excel at a diverse array of challenging tasks. The authors of this work use Q learning to approximate the state-action values of different states and the possible actions at each step in the game. There are many methods to solve this problem. Previously people have attempted to solve this problem using Dynamic programming, Monte Carlo methods etc. Such approaches work well for small state and action spaces but fail in the case of large state spaces and action spaces. In this paper the authors use a neural network to approximate the Q function. Neural networks are universal function approximators. Hence they're also able to learn the Q function. The main advantage of the neural network is its scalability. It can easily scale to large state space and action spaces.

In practice, neural networks are actually quite unstable for this task, since, even a small change to the weights of the neural network causes the agent's policy to change, which leads to drastically altered target values. This prevents the network from learning, and might also send the network into a feedback loop which causes the training process to become very unstable. Moreover, the data used to train the neural networks is highly correlated. This also has a destabilizing effect on the training process.

To address these challenges, the authors present two key ideas-

1. Experience replay
2. Use of a separate target network

2.1.1 Experience Replay

To perform experience replay, the authors store the agent's experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ at each time step t in a data set $D_t = e_1, ..., e_t$. During learning, mini batches of experiences are uniformly drawn at random from this dataset, which are then used to perform backpropogation and the weight updates. This breaks the correlation between data points in a batch.

2.1.2 Use of a separate target network

The Q function represents the expected reward reveived when starting from state s and taking action a.

$$Q^*(s,a) = \max_\pi \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...|s_t = s, a_t = a, \pi]$$

In order to approximate the optimal Q function, the neural network uses the following loss function-

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}[r + \gamma \max_{a'} Q(s', a'; \theta_i{}^-) - Q(s,a;\theta_i)]$$

Where θ_i are the parameters of the Q-network at iteration i and θ_i^- are the network parameters used to compute the target at iteration i. The target network parameters θ_i^- are only updated with the Q-network parameters θ_i every C steps and are held fixed between individual updates.

2.2 Evaluation

The authors evaluated their agent using Atari 2600 games. The same network architecture, hyperparameters and learning procedure was used throughout in order to demonstrate that the agent could learn successful policies over a variety of games. Out of the 49 Atari games that the Q-network was tested on, it outperformed the existing reinforcement learning methods on 43 of them without incorporating any additional prior knowledge that were used by the other methods. The DQN agent achieved over 75% of human score on 29 games.When the replay memory and the separate Q-network were not used, the DQNs performance dropped considerably.

In certain games, the DQN is also able to learn effective long term strategies. For example, in Breakout, the DQN learns the optimal strategy, which is to first dig a tunnel around the side of the wall allowing the ball to be sent around the back to destroy a large number of blocks.

The authors also used t-SNE to visualize the DQN's representation of states. The visualizations show that the DQN learns to map states that are close in terms of reward but perceptually dissimilar, close together. This observation is consistent with the notion that the network is able to learn representations that support adaptive behaviour from high-dimensional sensory inputs.

2.3 Shortcomings

Although the DQN shows impressive performance across a wide variety of games, there are certain games in which the DQN performs poorly. One of the major challenges for a DQN is learn temporally extended planning strategies, which are required in games such as Montezuma's Revenge. Moreover the DQN can't output continuous policies, which Policy Gradient based methods can do well.

2.4 Directions for future work

In future, the option of using DQNs to output a probability distribution over a continuous action space could be explored. Also, work needs to be done on incorporating memory into the network in order to make it capable of long term planning.